15-381 Artificial Intelligence, Fall 2008
Homework 4

Out: Oct 30, 2008

Due: Nov 13, 2008

Instructions

1 [12 pts] Local search

Give the name of the search algorithm that results from each of the following special cases:

1. Local beam search with $k = 1$

   Hill-climbing search

2. Local beam search with one initial state and no limit on the number of states retained

   Breadth-first search

3. Simulated annealing with $T = 0$ at all times (and omitting the termination test)

   Hill-climbing search (or first choice hill climbing described in the textbook)

4. Genetic algorithm with population size $N = 1$

   Hill-climbing search (or first choice hill climbing described in the textbook).

   Note that we always keep the best individual of the population, so the mutated individual only replace current individual when the fitness function improves. This is exactly first choice hill climbing described in the textbook and closest to hill climbing among the algorithms taught in the lectures.

2 [30 pts] Genetic algorithm

In this question, you need to propose crossover and mutation for genetic algorithms that result in "valid" states.

1. The 8-queens problem, as discussed in class, asks you to place 8 queens on an 8x8 chessboard such that no two queens can attack each other (i.e. share the same row, column, or diagonal). The chromosome (representing an individual) and crossover for the 8-queens problem shown in the class generate chromosomes with exactly one queen per column, but often more than one queens (or none) per row. Propose a chromosome representation and crossover and mutation mechanisms that generate result in exactly one queen per column and one queen per row.

   Hint: Look at the chromosome in the slides. What makes a chromosome valid (one queen per column and per row)? Think about how you can keep the chromosome valid. For crossover, you do not have to cut chromosome strings and concatenate them together. For mutation, you do not have to change one character in the chromosome strings.

A valid chromosome for 8-queens problem describing above can be represented by a permutation of integers from 1 to 8. There are several ways to crossover two permutations and here is one example:

For two permutations A and B, select a random crossover point after k, k between 1 to 7. For the first child, keep 1 to k from A and add other integers according to the ordering in B. For the second child, perform the same by exchanging the role of A and B.

For example, consider

```
A = 123 45678 \       / 123 75864
               X ==> X
B = 372 58614 /       \ 372 14568
```

For the first child, cut A into 123 and 45678, rearrange 45678 into 75864 according to B, resulting in the first child being 123+75864 = 12375864. Similarly the second child will be 372+14568 = 37214568.

2. The traveling salesman problem (TSP) is the problem of finding the shortest route to visit a set of cities exactly once and return to the starting city. In class, we showed that TSP can be solved by a genetic algorithm, but did not present the chromosome, crossover, and mutation. Propose a chromosome representation and the corresponding crossover and mutation, such that the chromosomes generated after crossover and mutation are still valid (visiting each city exactly once).

   Hint: It is likely that you can use a solution similar to your 8-queens solution. Think about what the valid chromosome representations for these two problems have in common?

   TSP of n cities can also be represented as a permutation of integers from 1 to $n-1$, assuming starting and ending at the n-th city. It is straightforward to extend the above method to permutations of arbitrary length.

3 [40 pts] Reinforcement learning

Consider reinforcement learning for the card game *blackjack*, or *twenty-one*. The following web page contains a description of the game and a demo of blackjack AI trained by reinforcement learning. You can experiment with this demo, or play blackjack against the dealer and trained AI.

`http://lslwww.epfl.ch/~anperez/BlackJack/classes/RLJavaBJ.html`

Note: Please do not run your experiments at the last moment, because we do not know if there is any download bottleneck on this server (even though applets run locally on your computer).

1. Propose a state space, including actions and rewards. Describing the actions in plain English is enough. There is no need to draw a transition diagram or table. Note that this is a simplified version of blackjack: The deck of 52 cards is shuffled before each hand. An Ace is automatically valued as 11, unless it would cause a bust. In that case it is valued as 1. Compare your state space to the real (unsimplified) blackjack game in the casinos, and describe any limitation of your state space.

   I will describe the actual state space used in the provided applet. The states basically correspond the score, the sum of card values in the player's hand. The score without an ace can be 4 to 20 (21 is a terminal state since one can only stand). The states with an ace is denoted as 23 to 31 (score plus 11). There are four terminal states, perfect, bust, win and lose. Hence the states are

$$\{4, 5, \cdots, 20, 23, 24, \cdots, 31, \text{perfect, bust, win, lose}\}$$

   Rewards are only defined for terminal states, R(perfect) = R(win) = 1, R(bust) = R(lose) = -1. Actions are hit or stand for non-terminal states.

There are a several limitations to this state space. First, no considerations for dealer's cards. Second, in real games cards are not shuffled before each hand, so considering cards already on the table may helps. Third, no considerations of special actions like double down, split a pair, etc.

2. Simulate Q-learning using your state space by hand for three epochs (three games). Set alpha to be 0.1, gamma to be 0.9, and epsilon to be 0.1. Start with a $Q(s, a)$ matrix initialized to all zeros except for terminal states s_{term}, which get $Q(s_{term}, a) = R(s_{term})$. When there is a tie, hit by default. Assume the following games (note that it is 6+Q now instead of 4+Q in previous version).

```
Game 1: start with 3 + 8, hit, given 6, hit, given 10, burst.
Game 2: start with 6 + Q, hit, given 2, hit, given 5, burst.
Game 3: start with J + 4, hit, given 2, stand, lose to the dealer.
```

Game 1
Start with $3 + 8 = 11$: hit
Given 6, $11 + 6 = 17$: no change, hit
Given 10, $17 + 10 = 27$: burst, $Q(17, h) = 0 + 0.1 \times (0 + 0.9 \times (-1) - 0) = -0.09$

Game 2
Start with $6 + Q = 16$: hit
Given 2, $16 + 2 = 18$: no change, hit
Given 5, $18 + 5 = 23$: burst, $Q(18, h) = 0 + 0.1 \times (0 + 0.9 \times (-1) - 0) = -0.09$

Game 3
Start with $J + 4 = 14$: hit
Given 2, $14 + 2 = 16$: stand but lose, $Q(16, s) = 0 + 0.1 \times (0 + 0.9 \times (-1) - 0) = -0.09$

3. Experiment the effect of changing the alpha (learning rate), gamma (discount factor), and epsilon (greediness) parameters on the demo program. Start with the default values, except set reward for win to be +1 and run 500 episodes instead of 1000 to save time. Modify one parameter at a time. Try a smaller value, a larger value, and the minimal and maximal values; report the winning rate of each parameter set (1 default and 12 modified parameter sets). Use 500 episodes instead of 1000 episodes to save time. What effects do these parameters have on the winning rate? Discuss potential explanation of these effects.

```
alpha  gamma  epsilon  win
 .01    .9     .01     41% (default)

 0      .9     .01     32%
 .25    .9     .01     40%
 .75    .9     .01     40%
 .99    .9     .01     40%

 .01    .01    .01     41%
 .01    .25    .01     42%
 .01    .75    .01     42%
 .01    .99    .01     42%

 .01    .9     .01     41%
 .01    .9     .25     39%
 .01    .9     .75     34%
 .01    .9     .99     32%
```

It is very difficult to significantly improves upon the default parameter set, but some values will significantly decrease the winning rate. Setting alpha to 0 means no learning, so performance are poor, close to random. Games are short so gamma does not make much difference. Setting epsilon to a small value does not hurt performance because blackjack is already very random. Setting epsilon to any moderately large value will hurt performance, reducing to random when gamma is close to 1, since the player usually takes a random action.

4. Describe how to use genetic algorithm for blackjack. Propose a chromosome representation, the crossover and mutation mechanisms, and the fitness function (or how to select the best individuals).

   There are 26 non-terminal states in the above state space. A chromosome representing a strategy can be a bit string of 0 and 1, 0 for stand and 1 for hit. Crossover is done by cutting two bit strings at a random location and switch the right hand side. Mutation is done by inverting any bit. To determine the fitness of the chromosomes, we can simulate the games for say 100 times. We can use the winning rate against a dealer by a predefined strategy, or even better, we can have all chromosome compete together while everyone take turns to be the dealer.

5. Discuss the advantages and disadvantages of Q-learning and genetic algorithm for blackjack.

   Most importantly, GA will take much longer time than Q-learning to achieve reasonable performance. Q-learning updates Q values towards the goal after every action, the only random part being ϵ-greedy which can be set to very rare. GA only updates strategy after many games, and there is a lot of randomness involved. On the other hand, Q-learning learns according to a fixed dealer. Potentially more than two different strategies can evolve and survive in GA (as in biological evolution), and other chromosomes can learn from them in the competition. For more complicated games Q-learning may be not as suitable while GA may do better, e.g. rewards are far more delayed, much more possible states and actions, etc. However the rules and consequences are simpler in blackjack, so it is likely not worth the time to run GA.

4 [18 pts] Property space

A man wishes to take a wolf, a goat and a cabbage across a broad river, from the south side to the north side, in a rowboat. The boat is very small and the poor man has room in the boat for only one of these items at a time. What's worse, he cannot leave the wolf and the goat on the same bank, whether coming or going, for the wolf will surely eat the goat. Nor can he leave the goat and cabbage on the same bank because the goat will eat the cabbage. He does not know what to do and turns to the smart students in 15-381 for help.

1. Describe this problem domain in a property space representation. Write the properties or literals that capture the problem space, the operators needed, and the operators' preconditions and postconditions.

   There are many possible ways to represent this domain in a property space representation. Here is one example. Note that for postconditions you cannot assume any mutual exclusive property, for example assuming WolfAt(b) implies not(WolfAt(a)). You have to state it explicitly as in the example below.

```
Move(a,b)
  preconditions: GoatAt(b) or CabbageAt(b), GoatAt(b) or WolfAt(b), ManAt(a)
  postconditions: not(ManAt(a)), ManAt(b)

MoveWolf(a,b)
  preconditions: WolfAt(a), GoatAt(b) or CabbageAt(b), ManAt(a)
  postconditions: not(WolfAt(a)), WolfAt(b), not(ManAt(a)), ManAt(b)

MoveGoat(a,b)
  preconditions: GoatAt(a), ManAt(a)
```

```
    postconditions: not(GoatAt(a)), GoatAt(b), not(ManAt(a)), ManAt(b)

  MoveCabbage(a,b)
    preconditions: CabbageAt(a), GoatAt(b) or WolfAt(b), ManAt(a)
    postconditions: not(CabbageAt(a)), CabbageAt(b), not(ManAt(a)), ManAt(b)
```

2. Write the initial state set I and goal state set G using your literals, and describe the solution path through property space by writing the operators and literals describing the journey from I to G.

```
  I: WolfAt(s), GoatAt(s), CabbageAt(s), ManAt(s),
     not(WolfAt(n)), not(GoatAt(n)), not(CabbageAt(n)), not(ManAt(n))
  G: WolfAt(n), GoatAt(n), CabbageAt(n)

  Solution path:
    MoveGoat(s,n)
    Move(n,s)
    MoveWolf(s,n)
    MoveGoat(n,s)
    MoveCabbage(s,n)
    Move(n,s)
    MoveGoat(s,n)
```